Structural Jacobian accumulation with unit edges

Andrew Lyons
Computation Institute, The University of Chicago
Mathematics and Computer Science Division, Argonne National Laboratory
lyonsam@gmail.com

Abstract

We consider the complexity of evaluating Jacobian matrices using a minimum number of floating point operations. We show that

We focus on the implications of the observation that linear operations result in partial derivatives that are unit values. On the edges in the graph that

The new variant of the optimal Jacobian accumulation problem presented here, which takes into account edges in the computational graph that have unit labels, occupies a middle ground between a very general problem that is known to be **NP**-hard and a subproblem whose complexity has been unknown for over two decades. We show that minimizing additions, multiplications, and total floating-point operations for this new variant are all **NP**-hard when using operations in $\{+, *\}$, even when subject to significant restrictions. We also present an example where the optimal accumulation over $\{+, -, *\}$ involves fewer multiplications than when subtraction is not allowed, providing the first evidence of the utility of subtraction in the accumulation of Jacobian matrices.

In fact, we find that the accumulation of Jacobian matrices forms a class of problems that includes many fundamental type of computations from linear algebra, including matrix products and in particular bilinear forms. We draw from results in the area of algebraic complexity theory to ...

We demonstrate that our results also apply to the evaluation of collections of Jacobian-vector and vector-Jacobian products.

1 Introduction

In Jacobian accumulation we are given a directed acyclic graph $G = (V, E)$, called a *computational graph*, which represents a particular (fixed) straight-line program that evaluates some vector function $F : \mathbb{R}^n \to \mathbb{R}^m$. The vertices in V represent the variables used in the evaluation of F; the edges $(u, v) \in E$ represent the direct dependencies among the variables and are labeled with the corresponding partial derivative, or *local partial*, of v with respect to u. Using the chain rule from derivative calculus, we may derive from the structure of G a collection of expressions over the edge labels, where each expression yields the value of one entry in the Jacobian matrix $J \equiv F'(\mathbf{x})$. We refer to the process of evaluating the entries of J via these expressions over the local partials as *Jacobian accumulation*, and our goal is to accumulate J with a minimum number of floating-point operations. The nature of the chain rule, and in particular the properties of the commutative ring $(\mathbb{R}, +, *)$, result in an exponential number of straight-line programs (or equivalently algebraic circuits) that accumulate J.

Jacobian accumulation arises in the field of automatic (or algorithmic) differentiation (AD) [GW08], which is a collection of techniques for obtaining derivatives for numerical programs. The process of determining an accumulation procedure is performed at compile time in the context of AD compilers for imperative languages. The problems presented here have direct applications and fall under the category of combinatorial scientific computing. AD tools such as Tapenade[1] and OpenAD[2] are used regularly to add derivatives to high-performance numerical codes such as the MIT General Circulation Model[3].

[1] http://www-sop.inria.fr/tropics/tapenade.html
[2] http://www.mcs.anl.gov/OpenAD/
[3] http://mitgcm.org/

compiled February 11, 2009 from draft version hg:ad8dfadd421d:50

This paper is organized as follows. In the remainder of this section we provide additional motivation, including an example, and give formal definitions for the problems we will discuss. In Section 3 we leverage a connection with the problem of optimal evaluation of bilinear forms to show that allowing subtractions in the accumulation process can lead to a reduction in the number of multiplications performed. In Section 2 we present complexity results for minimizing additions, subtractions, and total floating point operations for SOJA$_1$.

1.1 Jacobian matrices and Baur's formula

We consider vector functions of the form $\mathbf{y} = F(\mathbf{x}), F : \mathbb{R}^n \to \mathbb{R}^m$ that map a vector $\mathbf{x} = (x_i)_{i=1,\ldots,n}$ of *independent* variables to a vector $\mathbf{y} = (y_i)_{i=1,\ldots,m}$ of *dependent* variables. We assume that the way in which F is evaluated is fixed as a particular straight-line program, and we are given a directed acyclic graph (dag) G that is a computational graph for F. As noted above, the edges $(u, v) = e \in E$ are labeled with the local partials $c_e \equiv \frac{\partial v}{\partial u}$.

Baur's formula [BS83] yields the entries of the Jacobian as

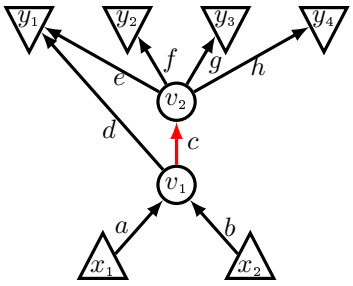
$$J_{j,i} \equiv \frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{[x_i \to y_j]}} \prod_{e \in P} c_e \quad ,$$

where $\mathcal{P}_{[x_i \to y_j]}$ denotes the set of all paths from x_i to y_j in G.

Example 1. Consider the vector function $\mathbf{y} = F(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}^4$ defined by $y_1 = (x_1 * x_2)/(x_1 * x_2 + 10)$, $y_2 = \sqrt{x_1 * x_2 + 10}$, $y_3 = \sin(x_1 * x_2 + 10)$, $y_4 = \cos(x_1 * x_2 + 10)$. The corresponding computational graph G is shown in Figure 1(b). Note that the values of the local partials may be trivially determined from the values that the variables in G assume during a particular evaluation of F: $a = \frac{\partial(x_1 * x_2)}{\partial x_1} = x_2$, $b = \frac{\partial(x_1 * x_2)}{\partial x_2} = x_1$, and so on. As such, we assume that an AD tool will automatically supply expressions for evaluating the local partials. Applying Baur's formula results in the following expressions for the entries of J: $J_{1,1} = ad + ace$, $J_{1,2} = bd + bce$, $J_{2,1} = acf$, $J_{2,2} = bcf$, $J_{3,1} = acg$, $J_{3,2} = bcg$, $J_{4,1} = ach$, $J_{4,2} = bch$.

$$\begin{aligned} v_1 &= x_1 * x_2 \\ v_2 &= v_1 + 10 \\ y_1 &= v_1 / v_2 \\ y_2 &= \sqrt{v_2} \\ y_3 &= \sin(v_2) \\ y_4 &= \cos(v_2) \end{aligned}$$

(a)



(b)

$$\begin{aligned} t_1 &= c * e; & t_2 &= d + t_1; \\ J_{1,1} &= a * t_2; & J_{1,2} &= b * t_2; \\ t_3 &= a * c; & t_4 &= b * c; \\ J_{2,1} &= t_3 * f; & J_{2,2} &= t_4 * f; \\ J_{3,1} &= t_3 * g; & J_{3,2} &= t_4 * g; \\ J_{4,1} &= t_3 * h; & J_{4,2} &= t_4 * h; \end{aligned}$$

(c)

Figure 1: Straight-line program for F (a), computational graph G (b), and a straight-line program that accumulates J (c) for Example 1.

Here and henceforth, unit edges are shown as red.

1.2 Optimal Jacobian accumulation

For many years researchers believed that the following problem (here stated as a decision problem), which addresses Jacobian accumulation through the purely *structural* aspects of G, appropriately captures the problem of accumulating J at minimum cost.

Problem 1. Structural Optimal Jacobian Accumulation (SOJA)
Instance: *Dag $G = (V, E)$, where each $e \in E$ is labeled with some c_e such that all c_e are unique real variables that are algebraically independent, positive integer K.*
Question: *Is there a straight-line program using operations in $\{+, *\}$ of length K or less that computes every entry in J such that every operand is either some c_e or the result of a previous operation?*

In practice, SOJA was first approached by means of a vertex elimination method [Yos87] similar to that used in Gaussian elimination on sparse matrices. Naumann introduced the progressively more general (and more powerful) techniques of *edge* [Nau99] and then *face* [Nau04] elimination. It is conjectured that for any instance of SOJA there is a face elimination sequence that accumulates the Jacobian at a minimum cost. However, these heuristic techniques do not directly apply (as currently defined) to the problems listed in the remainder of this section.

Recent work on the subject has highlighted the possibility of algebraic dependences among the local partials that label the edges of G. For example, the computational graph that corresponds to a matrix-vector product will have a great number of edges that are labeled with identical local partials. This realization has motivated the following more general problem.

Problem 2. Optimal Jacobian Accumulation (OJA)
Instance: *Dag $G = (V, E)$, where each $e \in E$ is labeled with some c_e that represents a real variable, positive integer K.*
Question: *Is there a straight-line program using operations in $\{+, *\}$ of length K or less that computes every entry in J such that every operand is either some c_e or the result of a previous operation?*

Theorem 1 ([Nau08])**.** OJA *is* **NP***-hard.*

The proof of this theorem by Naumann uses a reduction from the Ensemble Computation problem [GJ79] and relies heavily on the fact that there may be algebraic dependences among the edge labels. The result of this observation is that the complexity arises not strictly from the structure of the graph. In fact, the instances of OJA that are constructed by the proof consist of multiple disconnected components, each of which is a simple path. All entries of the resulting Jacobian lie on the diagonal.

All of the problems in this section have analogues where the goal is to minimize the number of additions or multiplications in the corresponding program. We denote these variants by a superscript $+$ or $*$, respectively. It may be argued that minimizing additions is of purely academic interest. However, minimizing multiplications has a practical significance, as they are significantly more costly than additions, and some computer architectures allow for a *used multiply-add* to be performed in a single clock cycle. Because the instances of OJA constructed in the proof of Theorem 1 result in expressions that have no additions, we immediately have the following result.

Corollary 1 ([Nau08])**.** OJA^* *is* **NP***-hard.*

Consider again the dag from Example 1, and note that $v_2 = v_1 + 10$. This implies that the edge label c will in fact always be equal to one, as $c \equiv \frac{\partial v_2}{\partial v_1} = 1$. Some or all of the edges in G may have such *positive unit labels*. With this as our motivation, we introduce the following new variant of Jacobian accumulation as a decision problem.

Problem 3. Structural OJA with Unit Edges (SOJA_1)
Instance: *Dag $G = (V, E)$, where each $e \in E$ is labeled with either a unique real variable c_e or a positive or negative unit label $+/-1$ such that all c_e are algebraically independent, positive integer K.*
Question: *Is there a straight-line program using operations in $\{+, *\}$ of length K or less that computes every entry in J such that every operand is either the label on some $e \in E$ or the result of a previous operation?*

As the definition of OJA does not have a specific provision for unit edges, we may similarly define Optimal Jacobian Accumulation with Unit Edges (OJA_1) as a superproblem of OJA where edges may have positive or negative unit labels $+/-1$. Theorem 1 and Corollary 1 apply immediately to OJA_1. Note that SOJA is a subproblem of SOJA_1, which in turn is a subproblem of OJA_1 (in other words, we have that $\text{SOJA} \subset \text{SOJA}_1 \subset \text{OJA}_1$). Our study of SOJA_1 is motivated by the fact that unit edges in the computational graph can be identified trivially, whereas recognizing algebraic dependences among the edge labels requires compiler analysis that is not performed by current AD tools.

The complexity results for SOJA_1 presented here are interesting for two reasons. First, they capture the complexity *inherent in the structure of G*, which will be necessary for any future complexity result for SOJA. Second, they show that SOJA_1 is **NP**-hard even under significant restrictions, thus taking a large chunk out of the problem space of Jacobian accumulation where we may find tractability. In particular, we show that SOJA_1 is **NP**-hard even when there is a bound of two on the indegree of all $v \in V$. This restriction

is motivated by the practical observation that AD compilers generate computational graphs from expression trees, which are often binary.

While it is clear that the number of additions, multiplications, or total operations in a straight-line Jacobian accumulation program can be verified in polynomial time, we must also verify that the program be algebraically equivalent to an evaluation of Baur's formula for the input dag G. Currently, no polynomial time procedure is known that can satisfy the latter requirement. Therefore, the problems stated here are not known to be in **NP**.

Though we speak of generating straight-line programs, we will occasionally represent these programs by expressions for the sake of brevity; the implied straight-line program should be derivable in a nonambiguous manner.

Note that the definitions of these problems restrict the arithmetic operations used in the accumulation procedure to those in $\{+, *\}$. In Section 3 we show that there are cases where the restriction to such *monotone* computations is restriction is not justified.

Key observations:

Observation 1.
- *We have a multiplicative identity (1)*
- *We have additive inverses (both a and $-a$)*

We will use Observation 1 to show that there are cases where the optimal nonmonotone accumulation circuit has fewer multiplications than the optimal monotone circuit.

Proposition 1. *For any binlinear form $\mathbf{a}^T R\mathbf{b}$ there exists a dag $G = (V, E)$ with unique minimal vertex x and unique minimal vertex y such that each edge $e \in E$ is labeled with some unique variable. Furthermore, we have that*

$$\sum_{P \in \mathcal{P}_{[x_i \to y_j]}} \prod_{e \in P} c = \mathbf{a}^T R\mathbf{b}.$$

Proposition 2. *For any binlinear form $\mathbf{a}^T R\mathbf{b}$ there exists a dag $G = (V, E)$ with unique minimal vertex x and unique minimal vertex y such that each edge $e \in E$ is labeled with some unique variable. Furthermore, we have that*

$$\sum_{P \in \mathcal{P}_{[x_i \to y_j]}} \prod_{e \in P} c = \mathbf{a}^T R\mathbf{b}.$$

In this section, we demonstrate that some fundamental problems from algebraic complexity can be modeled ... We begin by modeling the evaluation of bilinear forms as a Jacobian accumulation problem. This construction will be used later on to obtain complexity results for SOJA_1.

Let $\mathbf{a} = (a_1, a_2, \ldots, a_k)^T$ and $\mathbf{b} = (b_1, b_2, \ldots, b_\ell)^T$ be two vectors of indeterminates.

A formal definition of the class of functions we obtain is beyond the scope of this paper.

2 Complexity results

In this section we show that SOJA_1, SOJA_1^+, and SOJA_1^* are all **NP**-hard and remain so even under significant restrictions.

The complexity of SOJA_1^+ and SOJA_1 We will reduce Ensemble Computation to SOJA_1. An instance of Ensemble Computation [GJ79] consists of a finite set S, a collection $C = \{C_1, C_2, \ldots, C_r\}$ of distinct subsets of S, and a positive integer K. It is **NP**-hard to decide whether the sets in C can be constructed from the elements of S using K or fewer disjoint union operations.

Example 2. Consider the instance $S = \{a, b, c, d\}, C = \{\{a, b\}, \{a, c, d\}, \{b, c, d\}\}, K = 4$ of Ensemble Computation. The answer to this instance is YES, as all sets in C are yielded by the following collection of four union operations. $T = \{c\} \cup \{d\}$; $C_1 = \{a\} \cup \{b\}$; $C_2 = \{a\} \cup T$; $C_3 = \{b\} \cup T$. Figure 2(a) shows the corresponding instance of SOJA_1 that would be constructed as described in the proof of Lemma 1.

Lemma 1. SOJA_1^+ *is* **NP***-hard.*

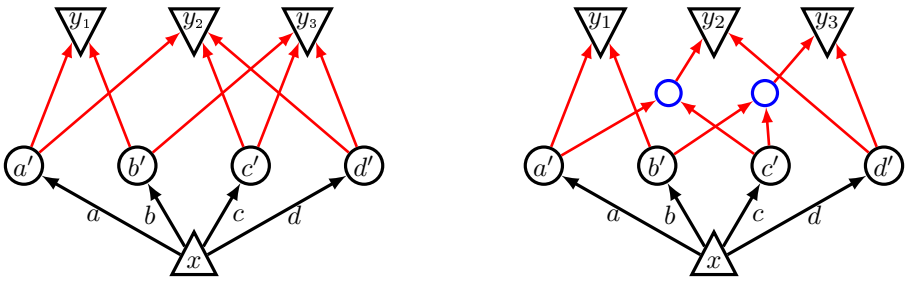


Figure 2: Instances of SOJA_1 that correspond to the instance of Ensemble Computation given in Example 2: all paths length ≤ 2 (a); all vertices indegree ≤ 2 and all paths length ≤ 3 (b).

Proof. Let S, C, K be an instance of Ensemble Computation. We construct an instance of SOJA_1 as follows, where Figure 2(a) depicts the construction for Example 2. Create a single independent vertex x, and for every $s \in S$, create a vertex s' and an edge from x to s' labeled s. For every set C_i create a dependent vertex y_i, and for all $s \in C_i$ create a unit edge from the corresponding vertex to y_i. We claim that the given instance of Ensemble Computation is a YES instance if and only if there is some straight-line program of length $\leq K$ that evaluates every nonzero entry of J.

Suppose all of the sets C_i can be constructed by some sequence of K or fewer disjoint union operations. We obtain from such a sequence a straight-line program for evaluating the entries of J by substituting the addition operation for disjoint union. There is a one-to-one correspondence between sets C_i and the nonzero Jacobian entries J_i, and it follows from Baur's formula that the disjoint unions that construct C_i will compute J_i when replaced by additions.

Suppose conversely that there is some straight-line program of length $\leq K$ that computes all nonzero entries of J. It follows from the nature of the constructed SOJA_1 instance G and the fact that we are restricted to operations in $\{+, *\}$ that such a program will involve additions exclusively. Furthermore, no operation in the program can produce a result that has more than one contribution from a particular nonunit edge label c_e, as no such label contributes more than once to any Jacobian entry and we are prohibited from using subtraction. We may therefore conclude that replacing the additions in the straight-line program with union operations will result in a sequence of $\leq K$ disjoint unions that constructs the sets in C.

We complete the proof by noting that the reduction described is indeed polynomial. □

The above construction will generate a graph with a single independent vertex, resulting in a Jacobian that represents a tangent. Obviously, a symmetric construction with a single dependent vertex and independent vertices for all $C_i \in C$ would yield the same result for gradients.

Theorem 2. SOJA_1 *is* **NP***-hard.*

Proof. The proof follows immediately from the fact that the instance of SOJA_1 that we construct as described in the proof of Lemma 1 involves additions exclusively. □

Note that the reduction described in the proof of Lemma 1 can result in vertices in G with $O(|V|)$ inedges. AD compilers often generate computational graphs with unary and binary operations exclusively, which results in a bound of two on the indegree of all $v \in V$.

Corollary 2. SOJA_1 *and* SOJA_1^+ *remain* **NP***-hard under each of the following restrictions.*

(i) G represents a tangent or gradient and all paths in G have length ≤ 2.

(ii) G represents a tangent or gradient, all vertices in G have indegree ≤ 2, and all paths in G have length ≤ 3.

Proof. (i) The proof follows directly from the proof of Theorem 1.

(ii) Since Ensemble Computation remains **NP**-hard even when all sets C_i satisfy $|C_i| \leq 3$, we may assume that all y_i in our constructed instance G have indegree ≤ 3. We create a new graph G' as follows,

where Figure 2(b) depicts the construction for Example 2. Let y_i be any such vertex with indegree 3, and let p_1, p_2, p_3 be the predecessors of y_i. Remove the unit edges (p_1, y_i) and (p_2, y_i), create a new vertex y_i', and create new unit edges (p_1, y_i'), (p_2, y_i'), and (y_i', y_i). It is clear that G' satisfies the conditions of (ii) and that the set of Jacobian expressions for G' yielded by Baur's formula are the same as those for G. Observe that no more than $|C|$ extra vertices are created, and thus the given transformation is polynomial. This completes the proof. □

Figure 2(b) shows the instance of SOJA_1 that corresponds to the instance of Ensemble Computation discussed in Example 2.

The complexity of SOJA_1^*. We will use the following problem in the reduction presented in this section.

Problem 4. Partition into Complete Bipartite Subgraphs
Instance: *Bipartite graph $G = (A, B, E)$, positive integer K.*
Question: *Can the edges of G be partitioned into $k \leq K$ disjoint sets $C_1, C_2, \ldots, C_k$ such that each C_i is a complete bipartite graph?*

Theorem 3 ([GJ80]). Partition into Complete Bipartite Subgraphs *is* **NP**-*complete.*

Theorem 4. SOJA_1^* *is* **NP**-*hard.*

Proof. We reduce Partition into Complete Bipartite Subgraphs to SOJA_1^*. Given an instance $G_B = (A, B, E_B), K$ of Partition into Complete Bipartite Subgraphs. Figure 3(b) depicts the following process for constructing an instance of SOJA_1^*. Orient every edge $\{a_i, b_j\} \in E_B$ from a_i to b_j, and give it the unit label 1. Add a new independent vertex x and a new dependent vertex y, then add edges labeled α_i from x to a_i for all $a_i \in A$ and edges labeled β_i from b_i to y for all $b_i \in B$. The resulting dag $G = (A \cup B \cup x \cup y, E_B \cup \{(x, a_i) \mid a_i \in A\} \cup \{(b_i, y) \mid b_i \in B\})$ will look something like the dag shown in Figure 3(b). We will show that the edges of G_B can be partitioned into K or fewer complete bipartite graphs (henceforth bicliques) if and only if the scalar Jacobian $J = (\partial y / \partial x)$ can be accumulated over $\{+, *\}$ using K or fewer multiplications.

Suppose some family of sets $C = \{C_1, \ldots, C_k\}$ partitions the edges of G_B into $k \leq K$ bicliques. We construct a Jacobian accumulation procedure using k multiplications as follows. Observe that each C_i comprises some $A_i \subseteq A$ and some $B_i \subseteq B$ and corresponds to some collection of paths in G. For each C_i, we first perform the necessary addition operations to compute the variables

$$\mathbf{z}_{A_i} = \sum_{a_j \in A_i} \alpha_j, \quad \mathbf{z}_{B_i} = \sum_{b_j \in B_i} \beta_j$$

followed by the multiplication $\mathbf{z}_i = \mathbf{z}_{A_i} * \mathbf{z}_{B_i}$. Finally, we add the necessary additions to compute $\mathbf{z}_J = \sum_{C_i} \mathbf{z}_i$.

Claim. $\mathbf{z}_J$, *computed as above, will yield the correct value for the Jacobian J.*

Proof of claim. Let $P_j^k \in \mathcal{P}_{[x \to y]}$ denote the unique path that passes through a_j and b_k. Observe that there is a one-to-one correspondence between unit edges (a_j, b_k) and the paths $P_j^k \in \mathcal{P}_{[x \to y]}$. It follows that the sets C_i partition $\mathcal{P}_{[x \to y]}$, and we thus have that

$$\mathbf{z}_J = \sum_{C_i \in C} \left(\sum_{a_j \in A_i} \alpha_j * \sum_{b_k \in B_i} \beta_k \right) = \sum_{P_j^k \in \mathcal{P}_{[x \to y]}} (\alpha_j * \beta_k) = \sum_{P \in \mathcal{P}_{[x \to y]}} \prod_{e \in P} c_e,$$

which completes the proof of the claim.

Suppose conversely that J can be computed by a straight-line program with operations in $\{+, *\}$ that uses K multiplications, and let $\mathbf{z}_i = \mathbf{z}_{A_i} * \mathbf{z}_{B_i}$ be some statement in the program. Because no two distinct $a_j, a_k \in A$ or $b_j, b_k \in B$ occur on the same path and divisions aren't allowed, both $\mathbf{z}_{A_i}$ and $\mathbf{z}_{B_i}$ must contain only contributions from inedges to vertices in $A_i \subseteq A$ and outedges of vertices in $B_i \subseteq B$, respectively, and

be constructed from addition operations exclusively. The subgraph C_i of G_B induced by A_i, B_i must be a biclique, as $\mathbf{z}_i$ would otherwise contain a contribution of the product of the labels on two edges that do not occur on a common path. Such a contribution cannot be negated, as we are restricted to operations in $\{+, *\}$. Thus $\mathbf{z}_i$ can contribute to the Jacobian only if A_i, B_i induce a biclique. Let $\mathbf{z}_i = \mathbf{z}_{A_i} * \mathbf{z}_{B_i}$, $\mathbf{z}_j = \mathbf{z}_{A_j} * \mathbf{z}_{B_j}$ be two distinct statements in the program. If both $\mathbf{z}_i$ and $\mathbf{z}_j$ contribute to the Jacobian, there can be no pair of vertices a_q, b_r such that $a_q \in A_i, A_j$ and $b_r \in B_i, B_j$, as this would result in two contributions of the product $\alpha_q \beta_r$ to the Jacobian, which cannot be negated by a subtraction or division. Because an edge in E_B corresponds to some path in G that contributes a product to the Jacobian, every such edge must be covered by some multiplication in the accumulation program. Therefore, we may conclude that the multiplications in the program correspond to a collection of K disjoint bicliques in G_B and that every edge in E_B is included in exactly one biclique, which completes our demonstration of a polynomial reduction from PARTITION INTO COMPLETE BIPARTITE SUBGRAPHS to SOJA$_1$. □

Corollary 3. SOJA$_1^*$ *remains* **NP**-*hard under each of the following restrictions.*

(i) G represents a scalar Jacobian and all paths in G have length ≤ 3.

(ii) G represents a scalar Jacobian and all vertices in G have indegree ≤ 2.

Proof. (i) This follows directly from the proof of Theorem 4.

(ii) Because of the nature of the construction described in the proof of Theorem 4, the only vertices in the resulting instance of SOJA$_1$ that can have indegree > 2 are those in B and y. Figure 3(c) depicts the following modification of our construction. If $|B| > 2$, replace the inedges of y with a binary tree rooted at y that consists of $|B| - 1$ dummy vertices, the outedges of which are all unit labeled. The edges labeled $\beta_1, \ldots, \beta_{|B|}$ that emanate from the vertices in B should now point to the minimal vertices in the new tree.

We handle the vertices in B in a similar manner. Let b be any vertex in B with indegree > 2, and let $A_b \subseteq A$ be the set of predecessors of b. Create a binary tree of $|A_b| - 1$ dummy vertices rooted at b that consists entirely of unit labeled edges. The leaves of the tree will be those vertices in A_b. Applying this process for each $b \in B$ results in the creation of $O(|B||A|)$ new vertices.

The process described above clearly maintains the property that the reduction from PARTITION INTO COMPLETE BIPARTITE SUBGRAPHS is polynomial. Furthermore, the graph G' that results from the above transformations satisfies the conditions of (ii) and the set of Jacobian expression for G' yielded by Baur's formula is the same as that for G. □

Example 3. The instance of SOJA$_1$ shown in Figure 3(a) would result from the instance of PARTITION INTO COMPLETE BIPARTITE SUBGRAPHS represented by the sets A, B of intermediate vertices. Figure 3(b) shows an instance where the indegree of each vertex is ≤ 2; it is equivalent to Figure 3(a) with respect to the expression obtained from Baur's formula. Accumulating J as $(\alpha_1 + \alpha_2)(\beta_1 + \beta_3) + (\alpha_1 + \alpha_3 + \alpha_4)\beta_2 + (\alpha_2 + \alpha_3 + \alpha_4)\beta_4$ carries a cost of three multiplications, which is the minimum for this example. This corresponds to optimally partitioning the underlying bipartite graph.

Minimizing Additions with Multiplications Fixed.

Theorem 5. *Given a particular set of multiplications to perform (as above) and a positive integer K, it is* **NP**-*complete to determine whether the values used as operands for the multiplications can be computed using K or fewer additions.*

Proof. The operands for the multiplications comprise two instances of *Ensemble Computation*.... □

For example, the optimal evaluation of the graph in Example 3 with respect to multiplications results in the following instances.

1. $S = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\},\ C = \{\{\alpha_1, \alpha_2\}, \{\alpha_1, \alpha_3, \alpha_4\}, \{\alpha_2, \alpha_3, \alpha_4\}\}$
2. $S = \{\beta_1, \beta_2, \beta_3, \beta_4\},\ C = \{\{\beta_1, \beta_3\}, \{\beta_2\}, \{\beta_4\}\}$

Note that the first instance is identical to the instance of *Ensemble Computation* in Example 2.

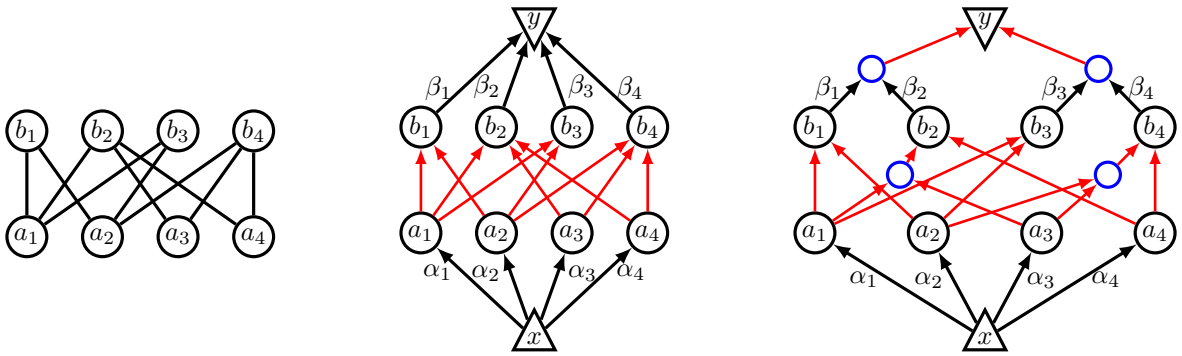


Figure 3: Instance of Partition into Complete Bipartite Subgraphs (a) and corresponding instances of SOJA_1 from Example 3: scalar, all paths length ≤ 3 (b); scalar, all vertices have indegree ≤ 2 (c).

3 The utility of subtraction

Note that the problems defined in Section 1.2 restrict the arithmetic operations used in the accumulation procedure to those in $\{+, *\}$. This restriction arose historically from the observation that Baur's formula involves additions and multiplications exclusively. In this section, we show that there are cases where this restriction is not justified.

Gonzalez and JáJá [GJ80] considered the optimal evaluation of bilinear forms $B = \boldsymbol{\alpha}^T R \boldsymbol{\beta}$, where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_p)^T$, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_q)^T$, and R is a $p \times q$ matrix whose elements are all in $\{0, 1\}$. As the following example demonstrates, any bilinear form $B = \boldsymbol{\alpha}^T R \boldsymbol{\beta}$ can be expressed as an instance of SOJA_1 such that accumulating the (scalar) Jacobian J yields the value of B.

The expression for the Jacobian yielded by Baur's formula for the type of computational graph constructed by the proof of Theorem 4 can be expressed as the bilinear form $\boldsymbol{\alpha}^T R \boldsymbol{\beta}$.

Example 4. The following bilinear form appears in [GJ80], where it is shown that B can be computed with only six multiplications using operations in $\{+, -, *\}$, whereas seven multiplications are required when using operations in $\{+, *\}$. Consider the graph shown in Figure 3(a).

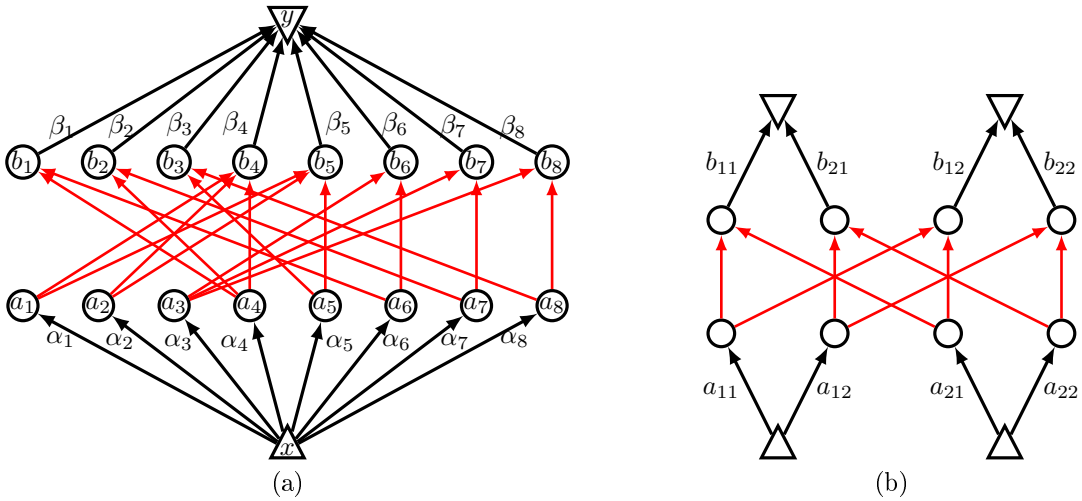
Baur's formula yields the following expression for J, which makes it equivalent to an example in [GJ80] of a bilinear form of the above type.

$$\begin{aligned} J &= \alpha_1\beta_4 + \alpha_1\beta_5 + \alpha_2\beta_4 + \alpha_2\beta_5 + \alpha_3\beta_6 + \alpha_3\beta_7 + \alpha_3\beta_8 + \alpha_4\beta_1 + \alpha_4\beta_2 \\ &\quad + \alpha_4\beta_4 + \alpha_5\beta_3 + \alpha_5\beta_5 + \alpha_6\beta_1 + \alpha_6\beta_6 + \alpha_7\beta_2 + \alpha_7\beta_7 + \alpha_8\beta_3 + \alpha_8\beta_8 \\ &= [\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4\ \alpha_5\ \alpha_6\ \alpha_7\ \alpha_8] \begin{bmatrix} \square & \square & \square & \blacksquare & \blacksquare & \square & \square & \square \\ \square & \square & \square & \blacksquare & \blacksquare & \square & \square & \square \\ \square & \square & \square & \square & \square & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \square & \blacksquare & \square & \square & \square & \square \\ \square & \square & \blacksquare & \square & \blacksquare & \square & \square & \square \\ \blacksquare & \square & \square & \square & \square & \blacksquare & \square & \square \\ \square & \blacksquare & \square & \square & \square & \square & \blacksquare & \square \\ \square & \square & \blacksquare & \square & \square & \square & \square & \blacksquare \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \\ \beta_7 \\ \beta_8 \end{bmatrix} \end{aligned}$$

Consider the graph shown in Figure 3, and note that Baur's formula yields the identical expression for J. Thus, because the expressions for B and J are identical, we may conclude that accumulating J requires seven multiplications when restricted to operations in $\{+, *\}$. In particular, J may be computed using operations in $\{+, -, *\}$ as

$$\begin{aligned} J &= (\alpha_1 + \alpha_2 + \alpha_3)(\beta_1 + \beta_2 + \beta_4) + (\alpha_3 + \alpha_6)(\beta_1 + \beta_6) + (\alpha_1 + \alpha_2 + \alpha_5)(\beta_3 + \beta_5) \\ &\quad + (\alpha_3 + \alpha_7)(\beta_2 + \beta_7) + (\alpha_3 + \alpha_8)(\beta_3 + \beta_8) - (\alpha_1 + \alpha_2 + \alpha_3)(\beta_1 + \beta_2 + \beta_3)\ . \end{aligned}$$

which requires only six multiplications.



(a) (b)

Figure 4: The accumulation of $F'(x)$ is cheaper when subtractions are allowed in the accumulation process. In (a)... In (b), a computational graph whose accumulation procedure represents the product of two 2×2 matrices. Strassen's algorithm demonstrates that **** admits an evaluation using only seven multiplications when subtraction is allowed.

Example 5 (Strassen's algorithm)**.** Strassen's algorithm [Str69] computes the product of two 2×2 matrices using only seven multiplications (whereas the niave algorithm uses eight multiplications).

$$\begin{aligned} J &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{11} + a_{12}b_{21} \\ a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{11} + a_{12}b_{21} \end{bmatrix} \end{aligned}$$

Both examples involve cases where commutativity can't help us []. But in our case, we aren't concerned with applying the algorithm recursively, so we could take advantage of the However, it has been shown that exploiting commutativity leads to only linear speedup over algorithms which do not [].

4 Conclusions and Open Problems

We have presented a formulation of optimal Jacobian accumulation that attempts to reflect the capabilities of modern AD tools as employed in the service of computational science. Problems in this area continue to provide interesting theoretical challenges. We are still searching for a problem definition for Jacobian accumulation that accurately captures the nature of the problem as it applies to practical applications of AD, and this search continues to provide interesting theoretical puzzles and connections to a broad array of areas within theoretical computer science. For example, Baur's formula bears a striking resemblance to a permanent. The complexity of SOJA, which has been unknown for more than two decades, remains an elusive and intriguing problem; polynomial time algorithms are known only for rather small classes of dags. Additionally, because Our main concern is the efficiency of the accumulation code, we may be willing to employ exponential time algorithms (at compile time) for solving Jacobian accumulation problems. This may be especially true for relatively small functions F that are executed in a loop body many times during the execution of a numerical code. Future research should focus on developing such algorithms to exploit unit edges, subtractions, and algebraic dependences between edge labels.

Acknowledgements. The author would like to thank Jean Utke and Uwe Naumann for numerous helpful discussions and suggestions.

References

[BS83] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–Completeness*. Freeman, San Francisco, 1979.

[GJ80] Teofilo F. Gonzalez and Joseph JáJá. On the complexity of computing bilinear forms with $\{0, 1\}$ constants. *J. Comput. Syst. Sci.*, 20(1):77–95, 1980.

[GW08] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 105 in Other Titles in Applied Mathematics. SIAM, Philadelphia, PA, 2nd edition, 2008.

[Nau99] Uwe Naumann. *Efficient Calculation of Jacobian Matrices by Optimized Application of the Chain Rule to Computational Graphs*. PhD thesis, Technical University of Dresden, December 1999.

[Nau04] Uwe Naumann. Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. *Mathematical Programming, Ser. A*, 99(3):399–421, 2004.

[Nau08] Uwe Naumann. Optimal Jacobian accumulation is NP-complete. *Mathematical Programming, Ser. A*, 112(2):427–441, 2008.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(3):354–356, 1969.

[Yos87] Toshinobu Yoshida. Derivation of a computational process for partial derivatives of functions using transformations of a graph. *Transactions of Information Processing Society of Japan*, 28(11):1112–1120, 1987.